The Good, the Bad and the Ugly

The Web Services Stack and Three Myths of Grids and Interoperability

Jon MacLaren

WSGA Workshop
Columbus, Ohio, Aug. 2006

Center for Computation & Technology
AT LOUISIANA STATE UNIVERSITY

What is Interoperability?

- **Webopedia Computer Dictionary:** The ability of software and hardware on different machines from different vendors to share data.
- Goal of interoperability is to get everyone working together, e.g. Grids
- So, if we write a “Grid Service” any (authorized) user should be able to use it. *(Technical - which I’ll deal with)*
- And maybe, that doing a given task, e.g. Job Submission, should always be the same everywhere. *(Social - which I won’t deal with)*

A Cause for Concern?

- Lots of concern about interoperability
- In the GGF/OGF, there is the “Grid Interoperation Now” effort to make large deployed Grids interoperate.
- Also, the WS-Interoperability Organization in the Web Services world (more later on this).
- It turns out that there are a lot of misconceptions about interop and about WS in general.

- I tackle three of these “myths”, and, after dispelling each one, I give advice which hopefully can provide the *methodological* basis for writing good, reliable interoperable services.

The Good

Using Web Services will
Give us Interoperability

WS will give us Interop

- See also: Free Lunches and Silver Bullets...
- Web Services *can* help us write interoperable software, but we still need to be careful

- Lets look at how people use the basic WS technology...

Problems with the tooling

- Much of the earlier tooling used WS to encode program interfaces, (think CORBA IDL)
- WSDL is generated from a code interface, e.g. Java2WSDL in Axis
- Consequently, people think of:
  - Services as objects
  - Operations as methods
  - Message parts as serialized arguments (objects/data structures)

Problems with the tooling

- Different toolsets encode things differently (no canonical encoding exists for Java)
- So to build clients to someone else's service, you must start with their generated WSDL
- No guarantee that another toolset can understand that encoding, e.g. that Axis can build a client to Websphere generated WSDL
- The odds get worse as you change language

Example Encodings Java2WSDL Java 1.5, Axis 1.4

- String
  - soapenc:string
- Vector<String>
  - <complexType name="Vector"><sequence>
    <element maxOccurs="unbounded" minOccurs="0”
    name="item" type="xsd:anyType"/>
    </sequence></complexType>
- String[]
  - <complexType name="ArrayOf_soapenc_string">
    <complexContent>
    <restriction base="soapenc:Array">
    <attribute ref="soapenc:arrayType”
    wsdl:arrayType="soapenc:string[]"/>
    </restriction>
    </complexContent>
    </complexType>

What Should You Do?

- **Go the other way:**
  - Think about the messages - design them
  - Write the XML Schema, and WSDL yourself, then generate the code interface from there
  - If you can't deal with the strange auto-generated classes for the messages, then turn them off and parse the DOM tree yourself

- This style is called "literal" as opposed to "encoded" (Literal is permitted by WS-I Basic Profile 1.x - Encoded is not!)
- Consequently, no soapenc types allowed...
- Now the WSDL is definitive (not a code interface), and building clients in other languages will be possible...

Who/What is WS-I?

- The WS-Interoperability Organization (WS-I) creates profiles which tell you what specs to use, but also **how** to use them
- A set of restrictions are defined...
- Complying to these helps you write interoperable web services.
  - ‘Specifically, WS-I creates, promotes and supports generic protocols for the interoperable exchange of messages between Web services. In this context, “generic protocols” are protocols that are independent of any action indicated by a message, other than those actions necessary for its secure, reliable and efficient delivery, and “interoperable” means suitable for multiple operating systems and multiple programming languages.’

What Shouldn't You Do?

- Some people can't be bothered with this, but they know what they want the XML to look like.
- So, they give in and write things like:
  - String processStockRequest(String xml)
- ...and parse the XML in the service code

- Please don't do this...

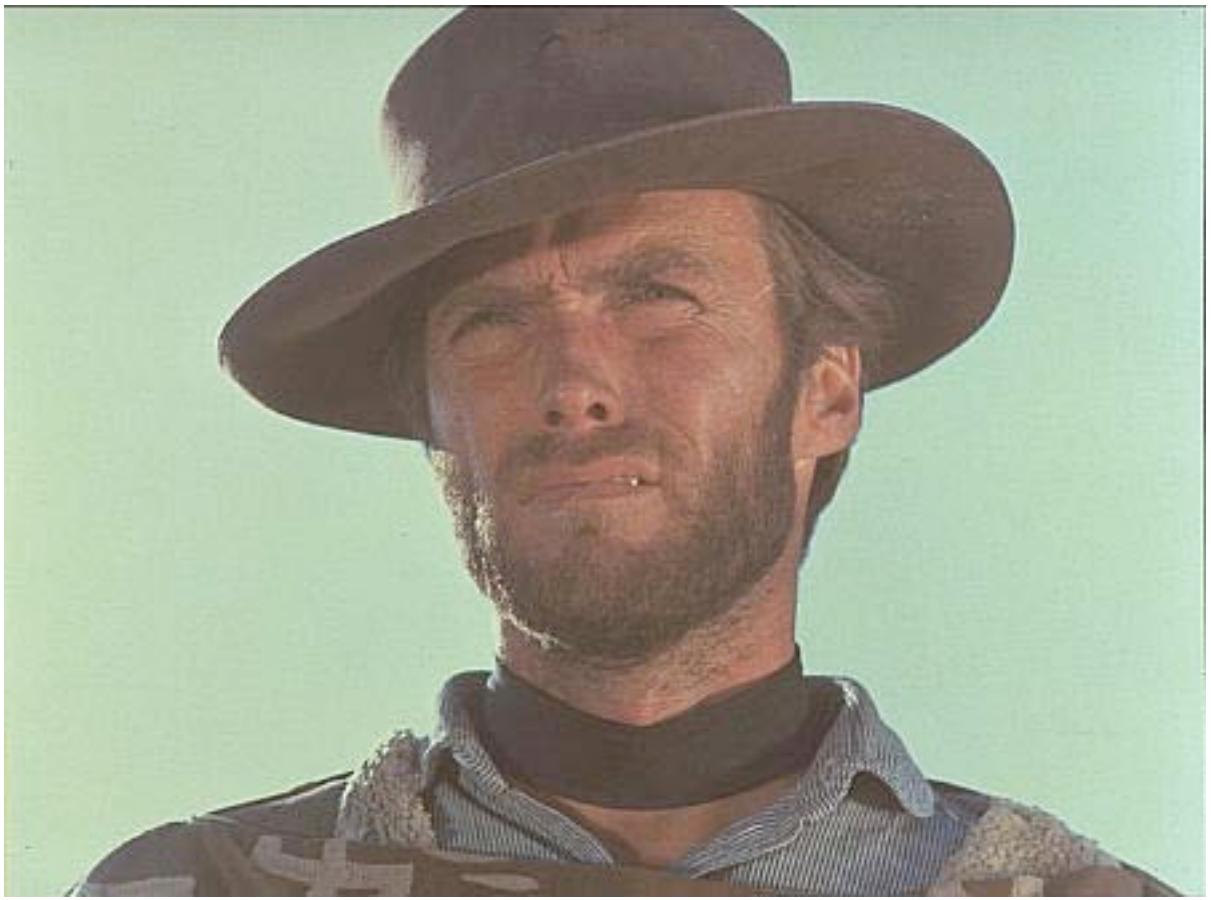
Other Advice

- Remember that in a distributed system, messages
  - get lost,
  - get delivered out-of-order, and
  - can be duplicated.

- If a client makes a request to your service, but don't get your reply, is there a way for them to find out what happened?
- Remote services can fail/become unreachable. Code using remote services must tolerate this sensibly.

- See: Waldo et al - "A Note on Distributed Computing"
- Also: MacLaren et al "Shelter from the Storm"

The Bad



Web Services are Stateless

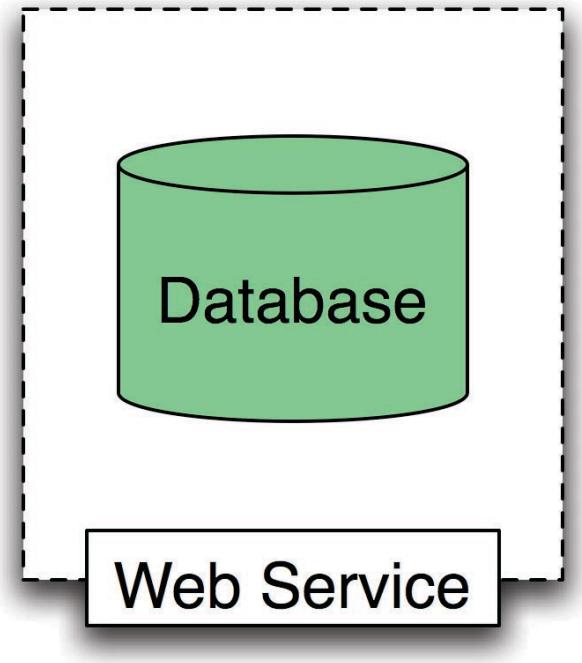
WS are Stateless

- Nonsense.
- Amazon sells books using WS - there's got to be some state *somewhere.*
- All useful services have state
- Possibly this comment was to do with "Stateless Protocols", where any message to the service can be understood by itself (it contains sufficient context, etc.)

- But, where do we *put* the state?

“Classic” Web Services

- Web Services are deployed at the organization’s boundary
- Provide (limited) ability for people outside to access/manipulate/interact with the organization’s internally-held state
- Thin layer. Stateless protocol.



- See: Helland, “Data on the Outside versus Data on the Inside”

Why Put It There?

- If you have a stateless protocol, and put all the state in a database, then:
  - You can deploy the service layer in a web farm
  - You get redundancy
  - You get failover
  - Without having to alter the service!

- Need to make the database reliable, but this is a pretty well understood problem

But if the service layer keeps some of the state?

- You can't sanely deploy on a web farm
- If the service dies, the state will be lost
- Oh. And this is better how?

- Of course, if you're using WS-RF your service can write all its state to a database
- But what about all the stuff that the container does? Are those EPRs in a database?

What Should You Do?

- My advice is to put all the state into a database, whether you are using WS-I or WS-RF.
- Need to at least be able to repair the service to the state it was in when it failed.
- (Can this be done with WS-RF?)
- Supporting multiple simultaneous instances of the same service, any of which can deal with the next request is harder (need to be able to synchronize via the database), but way cool
- That opens the door to highly-available services

The Ugly

Using the Same (Large) WS Stack Everywhere will give us Interoperability

Use Our Stack - Then Everything Will Be Fine...

- Lots of people are recommending complex "stacks" of Web Services specifications
- Often these contain emerging specifications, with no guarantee that:
  - they will be widely implemented
  - the implementations will interoperate well, or
  - that the standard will become widely used, or
  - the implementations will be maintained, etc.

- If these wishes don't come true, then interoperability is bound to be reduced, not increased (lock-in to obscure protocol).

OGSA

- OGSA-WG Providing Profiles, following the style of WS-I. Currently, there is a draft OGSA WSRF Basic Profile.
- It uses the following specs:
  - WS-Addressing
  - WS-ResourceProperties
  - WS-ResourceLifetime
  - WS-BaseNotification
  - WS-BaseFaults

OMII

- Open Middleware Infrastructure Institute, producing middleware for the UK e-Science Programme.
- From WSI Basic Profile:
  - XSD, WSDL, SOAP, UDDI
- From WSI Basic Security Profile:
  - WS-Security
- But then they add:
  - BPEL
  - WS-ReliableMessaging
  - WS-Addressing

When is a Standard Safe?

- Despite much confidence, WS-RF is going to fail. Why?
  - Because IBM said Yes, Microsoft said No.
- In March, a new roadmap for WS standards for “resources, events and management” was announced.
  - It covers the space occupied by WS-RF and WS-Notification
  - It’s backed by HP, IBM, Intel and Microsoft
- WS-RF will complete the OASIS process, but will never be widely adopted

Normal Rules Still Apply

- Just because we're doing Web Servces or Grid doesn't mean the usual principles suddenly don't apply
  - "Simple Systems Work and Complex Ones Don't" (Jim Gray)
  - KISS (Keep it Simple, Stupid!)
- Every WS spec you add increases the complexity of the system...

What Price Inclusion?

- For every component in the stack, you should ask/know:
  - "What does it give me that I need?"
- If you can't answer that, you probably don't need it.
- And don't forget the 80/20 rule - do you *really* need it?

- But if you do need the component, you also need to ask:
  - "What does this component cost me?"
- Cost may be to do with:
  - Responsiveness - does the stack gets slower?
  - Monetary - is there a free implementation?
  - Reliability - how solid is the implementation?
  - **Interoperability** - what platforms/languages are supported?

- It's a trade-off...

What Price Exclusion?

- But if you don't include the component, there can be another cost:
  - implementing the functionality you really *do* need that would have been provided
- However, you need to be careful here
- There are sometimes "End-to-end arguments" against pushing functionality down into the stack from the application

- See: Saltzer et al, "End-to-end Arguments in Systems Design"

What Should You Do?

- Keep the stack as simple as you can
- Be sure of why every part of you Web Services stack is there
- Fear the “bleeding edge”
- Don’t use emerging specs where possible
- (The same goes to spec writers!)

- And if you do get down as far as just using WS-I, then, why not go further...

Do you *really* need SOAP?

- How about XML over HTTP?
- Pro-WS people say SOAP can go over other things - not just HTTP
- But no-one ever uses this...
- HTTP is ubiquitous, well-supported in many languages

- For security, you can use:
  - HTTPS (Transport Level), or
  - XML Encryption/XML Signature—both W3C Recommendations—(Message Level)

I'm not alone...

"No matter how hard I try, I still think the WS-* stack is bloated, opaque, and insanely complex. I think it's going to be hard to understand, hard to implement, hard to interoperate, and hard to secure.

"I look at Google and Amazon and EBay and Salesforce and see them doing tens of millions of transactions a day involving pumping XML back and forth over HTTP, and I can't help noticing that they don't seem to need much WS-apparatus."

- Tim Bray, "The Loyal WS-Opposition"

Life Without SOAP

- I've written HTTP/XML services which you can write clients to in just about any language - even bash using curl (ugly!)
- No tooling required other than making/parsing XML, and making HTTP requests.
- Maybe you're not using the same stack everywhere - but who cares? You can still write a simple function that makes a request to my service, irrespective of whether your code is in some fancy container or not
- Don't need the same stack for services to talk to one another...
- This to me is maximizing interoperability...

- The other non-tech part (getting it adopted and deployed everywhere) is, of course, harder.

Are you a RESTafarian?

- REST (Representational State Transfer), by Roy Fielding also proposes a **lot** of other things
- This includes heavy usage of links (like HTML) to allow a client to navigate the content from a starting point
- I'm just using XML over HTTP - many people do this, and say they're doing REST
- I'm doing this because the stack is shorter, and it lowers the bar to using my services.

One Last Quote

- “Do we see that customers who develop applications using AWS care about REST or SOAP? Absolutely not! A small group of REST evangelists continue to use the Amazon Web Services numbers to drive that distinction, but we find that developers really just want to build their applications using the easiest toolkit they can find. They are not interested in what goes on the wire or how request URLs get constructed; they just want to build their applications.”

- Werner Vogels (Amazon CTO), interviewed by Jim Gray

Conclusions

- For interop you need good methodology, not just technology!
  1. Work out from the messages to the code
  2. Watch where you put the state
  3. Keep the stack simple
- Don’t propagate myths! Challenge the definitions/conclusions that you see others blindly accepting...
- Make up your own mind!

Questions?

References

- The Web Services-Interoperability Organization (WS-I), “WS-I Basic Profile 1.1”, http://www.ws-i.org/Profiles/BasicProfile-1.1.html
- Waldo et al, “A Note on Distributed Computing”, http://research.sun.com/techrep/1994/abstract-29.html
- MacLaren et al, “Shelter from the Storm: Building a Safe Archive in a Hostile World”, http://scoop.sura.org/documents/gada05.pdf
- Helland, “Data on the Outside versus Data on the Inside”, http://www-db.cs.wisc.edu/cidr/cidr2005/papers/P12.pdf
- OGSA-WG, OGSA WSRF Basic Profile, Currently at: https://forge.gridforum.org/sf/go/doc13542?nav=1
- OMII, “Web Service Grids: An Evolutionary Approach”, http://www.omii.ac.uk/dissemination/web_service_grids.jsp

References (cont.)

- HP/IBM/Intel/Microsoft, “Toward Converging Web Service Standards for Resources, Events, and Management”, http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/Harmonization_Roadmap.pdf
- Gray and Reuter, Transaction Processing, S3.8
- Saltzer et al, “End-to-end arguments in system design”, http://web.mit.edu/Saltzer/www/publications/endtoend/endtoend.pdf
- Tim Bray, “The Loyal WS-Opposition”, http://www.tbray.org/ongoing/When/200x/2004/09/18/WS-Oppo
- Fielding, “Architectural Styles and the Design of Network-based Software Architectures”, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- Werner Vogels, interviewed by Jim Gray, http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=388